APPLICATION PERFORMANCE MANAGEMENT BEST PRACTICES

Proactive Performance Engineering and DevOps

A collection of articles from the Compuware APM Center of Excellence

Compuware APM

Compuware

THE TECHNOLOGY PERFORMANCE COMPANY

WELCOME

When we're talking with prospective customers, perhaps the number one thing that we hear from them is "How do we get more proactive? How do we address performance problems before they impact end users?"

The answer has two components: firstly, taking a view of an application that spans its entire life-cycle, and building for performance from the earliest stages of creating the application. Secondly, breaking down the silos that exist between traditional stakeholders of the various phases of the lifecycle using approaches such as Agile or continuous integration on the pre-production side; or extending these ideas all the way through to deployment with approaches like DevOps. These approaches must be supported with cultural change, and an effective Application Performance Management solution.

Achieving consistently high levels of application performance and scalability requires a proactive approach to managing performance early in the application development lifecycle before new code is released to production. New approaches like DevOps are increasingly being adopted as modern frameworks for enterprise applications to be architected, developed and run. This volume explores embracing the DevOps culture and methodology to create and operate applications that deliver high-quality user experiences throughout the application lifecycle.

Demonstrating technology leadership, some organizations such as Thompson Reuters and Swarovski are already doing this. Many others have adopted some but not all elements of modern software creation and deployment, constrained by performance requirements.

I would like to thank all the authors who contributed to this publication, and you for your continued readership. You can find our latest and archived articles at http://apmblog.compuware.com.

Andreas Grabner

Compuware APM Center of Excellence

Proactive Performance Engineering & DevOps

IT TAKES MORE THAN A TOOL! SWAROVSKI'S 10 REQUIREMENTS FOR CREATING AN APM CULTURE

— Andreas Grabner, November 21, 2012

Swarovski—the leading producer of cut crystal in the world—relies on its ecommerce store, much like other companies do in the highly competitive ecommerce environment. Swarovski's story is no different from others in this space: it started with "Let's build a web site to sell our products online" a couple of years ago and quickly progressed to "We sell to 60 million annual visitors across 23 countries in 6 languages." There were bumps along the road and they realized that it takes more than just a bunch of servers and tools to keep the site running.

WHY USE APM INSTEAD OF JUST A TOOL?

Swarovski relies on Intershop's ecommerce platform and faced several challenges as the ecommerce business grew rapidly. Those challenges required the company to apply application performance management (APM) practices to ensure it could fulfill the business requirements to keep pace with customer growth while maintaining an excellent user experience. The most insightful comment I heard was from René Neubacher, Senior eBusiness Technology Consultant at Swarovski: *"APM is not just about software. APM is a culture, a mindset, and a set of business processes. APM software supports that."*

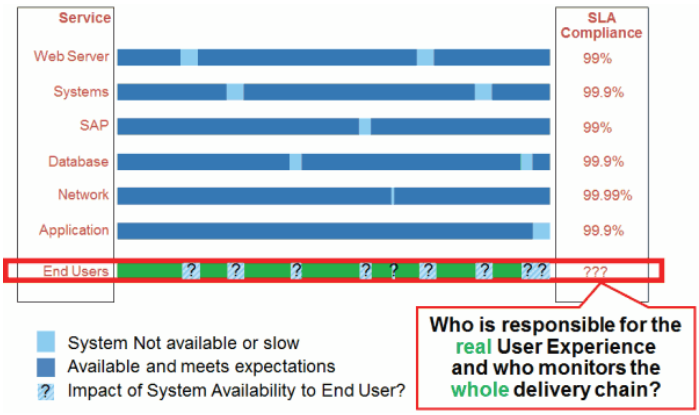
René recently discussed Swarovski's journey to APM, what the initial problems were, and what requirements the company ended up having for APM and the tools needed to support Swarovski's APM strategy. The company reached the next level of maturity by establishing a Performance Center of Excellence that tackles application performance proactively throughout the organization instead of putting out fires reactively in production.

This blog post describes the challenges Swarovski faced, the questions that arose, and the new generation of APM requirements that paved the way forward.

THE CHALLENGE

Swarovski had traditional system monitoring in place on all its systems across the delivery chain, including web servers, application servers, SAP, database servers, external systems, and the network. Knowing that each individual component is up and running 99.99% of the time is great, but no longer sufficient.

How might individual component outages impact the user experience of online shoppers? *Who* is actually responsible for the end user experience and *how* should you monitor the complete delivery chain and not just the individual components? These and other questions came up when the ecommerce site attracted more customers, quickly followed by more complaints about the user experience:



APM includes getting a holistic view of the complete delivery chain and requires someone to be responsible for end-user experience.

QUESTIONS THAT NEEDED ANSWERS

In addition to *Who is responsible in case users complain?* the other questions that needed to be urgently addressed included the following:

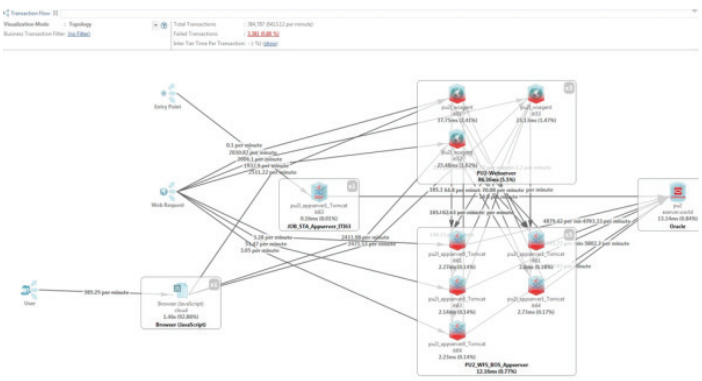
- How often is the service desk called before IT knows that there is a problem?
- How much time is spent in searching for system errors versus building new features?
- Do we have a process to find the root cause when a customer reports a problem?
- How do we visualize our services from the customer's point of view?
- How much revenue, brand image, and productivity are at risk or lost while IT is searching for the problem?
- What should be done when someone says "it's slow"?

THE 10 REQUIREMENTS

These questions triggered the need to move away from traditional system monitoring and develop the requirements for new-generation APM and user-experience management.

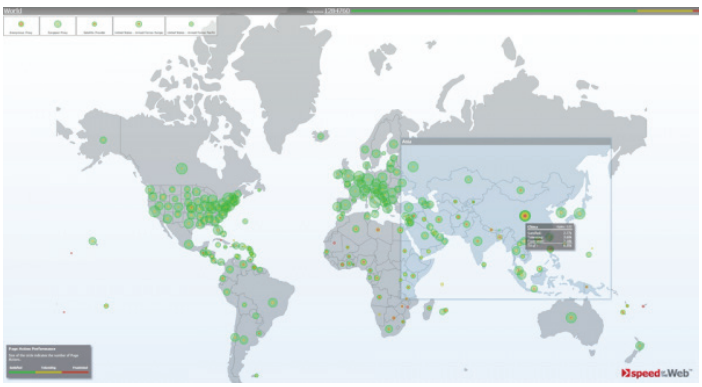
#1: SUPPORT STATE-OF-THE-ART ARCHITECTURE

Swarovski needed an approach that would work with its system architecture, now and in the future. The increase in interactive Web 2.0 and mobile applications had to be factored in to allow monitoring of end users from many different devices and regardless of whether they used a web application or mobile-native application as their access point.



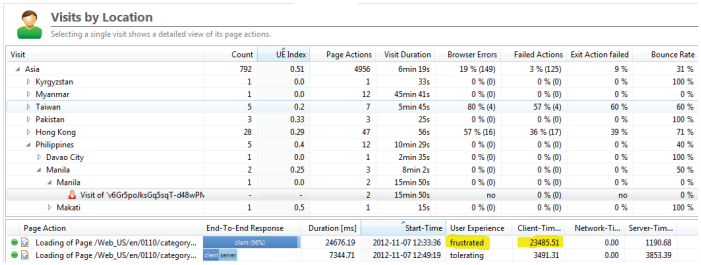
Transactions need to be followed from the browser all the way back to the database. It is important to support distributed transactions. This approach also helps to spot architectural and deployment problems immediately.

#2: 100% TRANSACTIONS AND CLICKS—NO AVERAGES



Measuring end-user performance of every customer interaction allows for quick identification of regional problems with CDNs, third parties, or latency.

Based on experience, Swarovski knew that looking at average values or sampled data would not be helpful when customers complained about bad performance. Responding to a customer complaint with "Our average user has no problem right now—sorry for your inconvenience" is not what you want your helpdesk engineers to do. Averages and sampling also hide the real problems in your system. Check out the blog post *Why Averages Suck*, **http://apm-blog.compuware.com/2012/11/14/why-averages-suck-and-percentiles-are-great/**, by Michael Kopp, for more detail.



Having 100% user interactions and transactions available makes it easy to identify the root cause of problems for individual users.

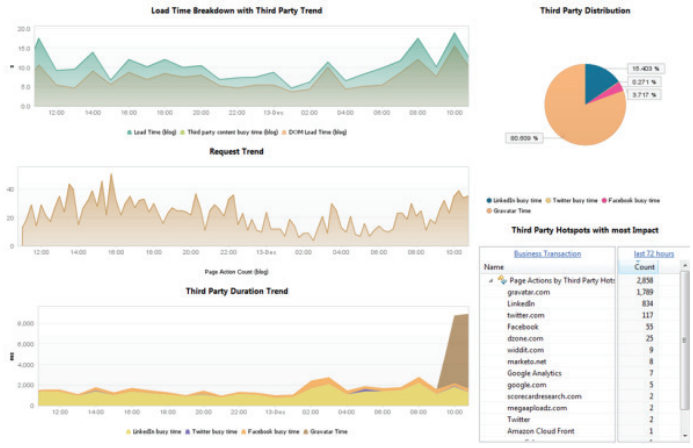
#3: BUSINESS VISIBILITY

As the business had a growing interest in the success of the ecommerce platform, IT had to demonstrate to the business what it took to fulfill the requirements and how business requirements are impacted by the availability or the lack of investment in the application delivery chain.

Correlating the number of visits with performance on incoming orders illustrates the measurable impact of performance on revenue and what it takes to support business requirements.

#4: IMPACT OF THIRD PARTIES AND CDNS

It was important to track not only transactions involving Swarovski's own data center, but *all* user interactions with the company's web site—even those delivered through content-delivery networks (CDNs) or third parties. All of these interactions make up the user experience and therefore *all* of them need to be analyzed.



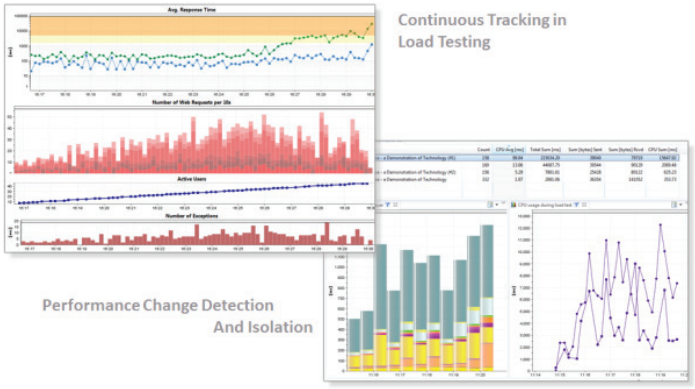
Seeing the actual load impact of third-party components or content delivered from CDNs enables IT to pinpoint user-experience problems that originate outside the company's own data center.

#5: ACROSS THE LIFECYCLE—SUPPORTING COLLABORATION AND TEARING DOWN SILOS

Continuously catching regressions in development by analyzing unit and performance tests allows application teams to become more proactive.

The APM initiative was started because Swarovski reacted to problems happening in production. Fixing these problems in production is only the first step. The ultimate goal is to become proactive by finding and fixing problems in development or testing—before they spill over into production.

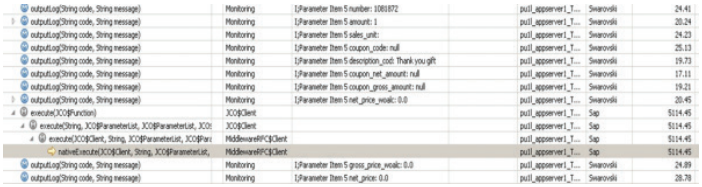
Instead of relying on different sets of tools with different capabilities, the requirement is to use one single solution that is designed to be used across the application lifecycle (developer workstation, continuous integration, testing, staging, and production). It will make it easier to share application performance data between lifecycle stages, allowing individuals to not only easily look at data from other stages, but also compare data to verify the impact and behavior of code changes between version updates.



Pinpointing integration and scalability issues, continuously, in acceptance and load testing makes testing more efficient and prevents problems from reaching production.

#6: DOWN TO THE SOURCE CODE

In order to speed up problem resolution, Swarovski's operations and development teams require as much code-level insight as possible—not only for the company's own engineers who are extending the Intershop ecommerce platform, but also for Intershop to improve its product. Knowing what part of the application code is not performing well with certain input parameters or under a specific load on the system eliminates tedious reproduction of the problem. The requirement is to lower the mean time to repair (MTTR) from as much as several days down to only a couple of hours.



The SAP connector turned out to have a performance problem. This method-level detailed information was captured without changing any code.

#7: ZERO/ACCEPTABLE OVERHEAD

"Who are we kidding? There is nothing like zero overhead, especially when you need 100% coverage!" These were René Neubacher's words when he caught wind of that requirement. And he is right: once you start collecting information from a production system you add a certain amount of overhead. A better term than "zero overhead" would be "imperceptible overhead"—overhead that's so small, you don't notice it.

What is the exact number? It depends on your business and your users. The number should be worked out from the impact on the end-user experience rather than additional CPU, memory, or network bandwidth required in the data center. Swarovski knew it had to achieve less than 2% overhead on page load times in production, as anything more would have hurt its business.

#8: CENTRALIZED DATA COLLECTION AND ADMINISTRATION

Running a distributed ecommerce application that gets potentially extended to additional geographical locations requires an APM system with a centralized data-collection and administration option. It is not feasible to collect different types of performance information from different systems, servers, or even data centers. It would either require multiple analysis tools or data transformation to a single format to use it for proper analysis.

Instead of this approach, Swarovski needed a single unified APM system. Central administration was equally important, as the company needed to eliminate the reliance on remote IT administrators to make changes to the monitored system—for example, simple tasks such as changing the level of captured data or upgrading to a new version.

Storing and accessing performance data from a single, centralized repository enables fast and powerful analysis and visualization. For example, system metrics such as CPU utilization can be correlated with end-user response time or database execution time—all displayed on a single dashboard.

#9: AUTO-ADAPTING INSTRUMENTATION WITHOUT DIGGING THROUGH CODE

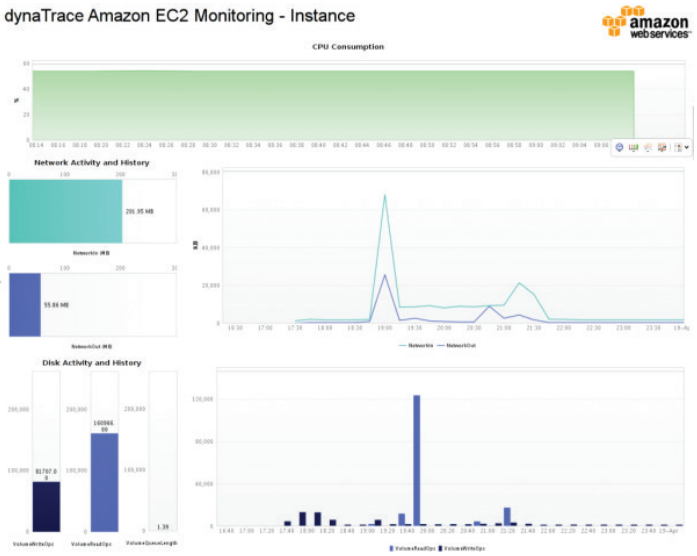
As the majority of Swarovski's application code is not developed in-house but provided by Intershop, it is mandatory to get insight into the application without doing any manual code changes. The APM system must auto-adapt to changes so that no manual configuration change is necessary when a new version of the application is deployed.

This means Swarovski can focus on making its applications contribute positively to business outcomes rather than spend time maintaining IT systems.

#10: ABILITY TO EXTEND

Swarovski's application is an always-growing and ever-changing IT environment. Elements that may have been deployed on physical boxes might be moved to virtualized environments or even into a public cloud environment.

Whatever the circumstance, the APM solution must be able to adapt to these changes and be extensible to consume new types of data sources—e.g., performance metrics from Amazon Cloud Services or VMware, Cassandra or other big-data solutions, or even extend to legacy mainframe applications and then bring these metrics into the centralized data repository and provide new insights into the application's performance.



Extending the application-monitoring capabilities to Amazon EC2, Microsoft Windows Azure, or another public or private cloud enables the analysis of the performance impact of these virtualized environments on end-user experience.

THE SOLUTION AND THE WAY FORWARD

Swarovski took the first step in implementing APM as a new process and mindset in the organization. The company is now in the next phase of implementing a Performance Center of Excellence. This allows movement from reactive performance troubleshooting to proactive performance prevention.

Stay tuned for more blog posts on the Performance Center of Excellence and how you can build one in your own organization. The key message is that it is not about just using a bunch of tools. It is about living and breathing performance throughout the organization. If you are interested in this topic, check out these blogs by Steve Wilson:

Proactive vs. Reactive: How to Prevent Problems Instead of Fixing Them Faster: **http://apmblog.compuware.com/2012/07/12/proactive-or-reactive-a-guide-to-prevent-problems-instead-of-fixing-them-faster/**

Performance in Development Is the Chief Cornerstone: **http://apmblog.compuware.com/2012/09/05/performance-in-development-is-the-chief-cornerstone/**

WHO IS TO BLAME FOR BAD APPLICATION PERFORMANCE?

— Alois Reitbauer, May 28, 2011

When something goes wrong, who's to blame? Very often the question as to who is responsible is at least as important as how to solve the problem. In order to not have to search for the responsible person all the time, I aim to find a general rule on who is responsible for performance problems.

Let's start by looking at all the people who are involved in the software-delivery process and investigate whether each one is responsible for a performance problem.

THE DEVELOPER

Obviously the developer must be responsible, as he has written the code that is not performing well. OK—problem solved. But wait; this cannot be true. First of all, developers write well-performing code. They are very good at what they do. Also, all their testing showed that everything was working fine, so something must have gone wrong later in the application lifecycle. Furthermore, if it was developer error, it was not the developer's code, but rather another developer's code. Our developer has always had this gut feeling that the service he was calling somehow was slow. He would have built that application differently anyway—it was just this architect guy who decided it had to be that way.

THE ARCHITECT

So it is the architect who is responsible. He designed well-performing architecture that scales infinitely. However, it obviously does not. Clearly the reason is not the architecture, because it is undoubtedly great. It uses hyped technologies and the coolest framework for just about everything. So if he would have coded the application, everything would be perfect. He, however, just did not have the time to write the code along with everything else. As always, the developer did not get the architecture implemented properly. He has printed all those cool diagrams (even in color) and those devs screwed it up. Obviously the head of R&D cannot get his people to do the simplest thing on earth—build the application as it was designed.

THE HEAD OF R&D

The head of R&D screwed it up. He had everything he needed, but he did not get the job done. He could not manage his developers to get their jobs done. We even sent him to a Scrum seminar—a pretty expensive one—and this did not help. We shortened release cycles and got more flexible with requirements, and he still cannot get the job done. However, he does not feel responsible at all. He shipped a new iteration release every week. The testing guys needed weeks to provide feedback based on a code base that was everything but up-to-date. The feedback that "tests failed at 300 users" was not terribly helpful either. He had to task three developers with adding debug output and holding the hands of the testing people. In the end, his guys did the job better themselves. Who needs testers if the hard work is ultimately done by development?

THE TESTER

Obliviously the tester is responsible. How couldn't we see this from the beginning? He should test whether the application performs as expected. If we tested properly, there would not be any problems. Why do we give them all these expensive testing tools and hardware if the outcome is just something like "we have a problem"? The tester, however, does not feel responsible at all. He wrote all the test scripts and ran the tests. He does not know the code, so how should he know what went wrong? If the developers know exactly what kind of information they need, why didn't they simply put the proper tracing into the code? He then simply sends back the logs and they fix it—life should be so simple. Instead they come over and block the testing infrastructure to find the problem. In the end, it is the tester who gets blamed when he cannot finish testing in time. Ultimately everything worked out anyway—all the tests ran perfectly—so something must have gone wrong in production.

THE OPERATIONS GUY

Why didn't we search for the responsible party closest to where the problem occurs? All others got it right in the end, and then Operations is unable to run the application. All they have to do is keep everything up and running. We even wrote a whole run book for them. Where is the problem? Well, that's hard to know; and when there is a problem and you need information, they behave like getting logs and other data is a matter of national security. If we would just get access to that production system, we would fix it right away. Talking to the Operations guy draws a different picture. All he is doing is running the application as he was told. If the development team gives Operations the wrong information, it is not their fault. They try to keep the service available by all means possible. Even more important, they have not written the code and they are damn sure that this is an application and not an infrastructure problem. And yes … they love to send around log files, change debug levels, and restart the server every couple of hours. That's why they started to work in IT and this is also their ultimate destiny. So if the developers would just …

… wait! It seems like we are getting back to where we started. At the end everybody is and is not responsible at the same time. Nobody did anything wrong, but there are reasons for blaming everyone else for why things did not work out.

CONCLUSION

What is the conclusion? Well, following in the footsteps of the famous German writer Bertolt Brecht, there is no conclusion. It is up to the audience—you—to make up your mind. However, like always, finding the responsible person won't solve any problems.

TOP PERFORMANCE MISTAKES WHEN MOVING FROM TEST TO PRODUCTION: EXCESSIVE LOGGING

— Andreas Grabner, November 1, 2012

When things work great in testing it doesn't necessarily mean that everything works great in production. There are many things that might be different, such as much higher load than tested, different hardware, external services that couldn't be tested, and real users that execute transactions that weren't tested.

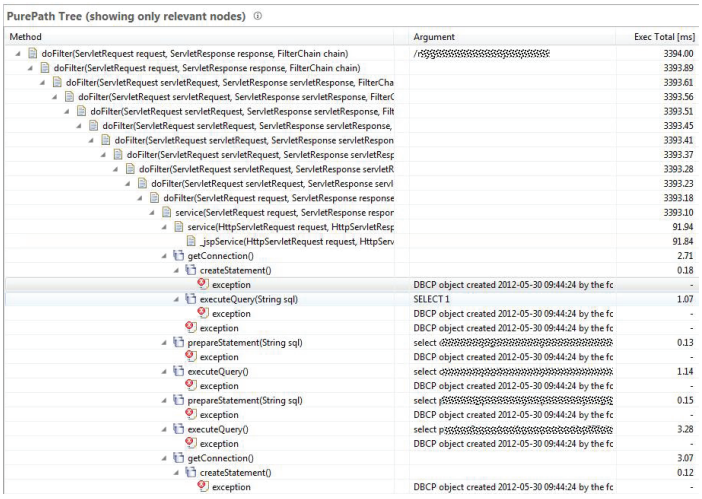
This blog post marks the start of a series covering the top performance mistakes when moving from testing to production. All examples are taken from real-life applications we have been working with. I will start the series with a topic that many application owners may not think about: excessive logging.

LOGGING IS IMPORTANT—BUT HANDLE WITH CARE

We all know that logging is important, as it provides valuable information when it comes to troubleshooting. Logging frameworks make it easy to generate log entries and, with the help of tools, it's becoming easier to consume and analyze log files when it is time to troubleshoot a problem. But what if there is too much log information that nobody looks at, that just fills up your file system and—worse—seriously impacts your application performance and thereby introduces a new problem to you application. The following two examples are taken from an online web application running on Tomcat. You can easily see how the current logging settings can become a real problem once this application gets deployed.

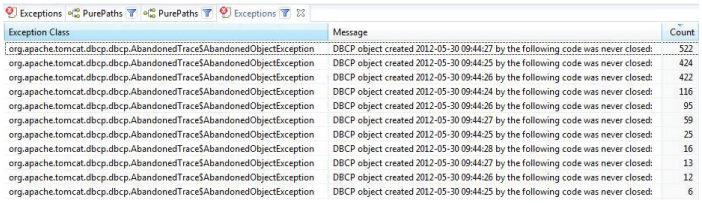
EXAMPLE #1: EXCESSIVE USE OF EXCEPTION OBJECTS TO LOG STACK TRACE

In this example the Tomcat connection pool was used with the setting "logAbandoned=true." Here the connection pool runs into a problem; it starts logging a lot of information, including full stack traces of where the application tried to get a connection from the pool. A typical transaction executed several hundred database queries. Most queries were executed on separate connections, which resulted in about as many getConnection calls as executedQuery calls.



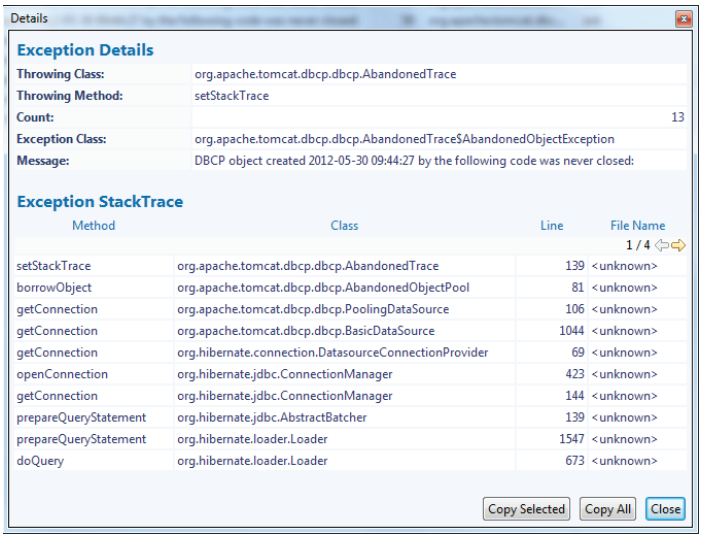
High overhead when creating hundreds of exception objects to get full stack-trace information

Looking just at the exception objects gives us a clearer picture of how much information gets collected but won't provide a whole lot of useful information.



Several hundred exception objects are created just for a single web request.

When looking at the exception details of these individual exception objects, it is easy to see which class actually creates these objects in order to obtain the stack-trace information for logging.



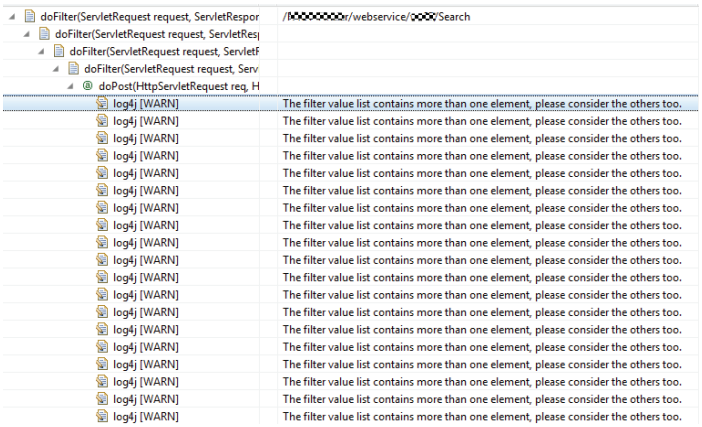
The AbandonedTrace.setStackTrace method was used to create these exception objects.

ADVICE FOR PRODUCTION

In the test environment this might not be a big issue if you're only simulating a fraction of the load expected on the live production system. It must, however, be a warning to the one responsible for moving this application to production. More information on this particular problem can be found here: **https://groups.google.com/forum/?fromgroups#!topic/mmbase-discuss/5x24EjBZMGA**.

EXAMPLE #2: TOO MUCH AND TOO GRANULAR USE OF LOGGING

Many applications have very detailed logging in order to make problem analysis easier in development and testing. When deploying an application in production people often forget to turn off DEBUG, FINE, FINEST, or INFO logs, which would flood the file system with useless log information and cause additional runtime overhead. It also often happens that log levels are used incorrectly, which leads to too many WARNING or ERROR logs for matters that are not a problem at all. This will make it very hard when analyzing log files to find the real problems. The following image shows a log message writing multiple times in the same transaction using the wrong log level.



Excessive logging and incorrect usage of severity level results in too much time spent analyzing log output.

ADVICE FOR PRODUCTION

- Make sure you configure your log levels correctly when deploying to production.
- Test these settings in your test environment and verify that there are no DEBUG, FINE, or other such log messages still logged out to the file system.
- Show the generated log files to the people that will have to use these log files in case of a production problem, and get their agreement that these log entries make sense and don't contain duplicated or "spam" messages.

ADDITIONAL READING

If you are interested in load testing, check out the post *To Load-Test or Not to Load-Test: That Is Not the Question*: **http://apmblog.compuware.com/2011/09/28/to-load-test-or-not-to-load-test-that-is-not-the-question/**

PERFORMANCE IS THE CORNERSTONE OF DEVELOPMENT

— Steve Wilson, September 5, 2012

In Roman architecture the stone a construction project was started from was called the cornerstone or foundation stone. It was the most important because this was the stone that the placement of all other stones would be based on. Great care was taken in making sure that the angles were correct. If there was a slight deviation, even one degree, it could cause structural issues for a foundation. When talking about performance across the lifecycle we must start at the foundation. In another post, **http://apmblog.compuware.com/2012/07/12/proactive-or-reactive-a-guide-to-prevent-problems-instead-of-fixing-them-faster/**, I talk about the need for looking at performance not just in a single phase but in all aspects of the lifecycle. In this article we will look at the need for performance to be the cornerstone of the development process.

WHAT IS THE CORNERSTONE FOR MOST DEVELOPMENT TEAMS?

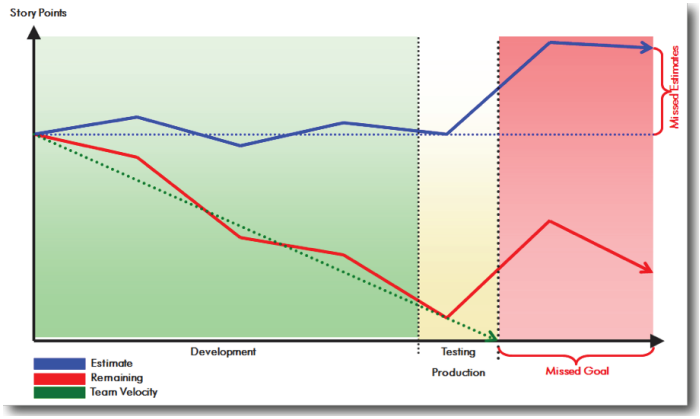
When researching what the top priorities for development teams are I was surprised to find that performance was rarely mentioned. Focus was typically on other issues—things like the scope of an application, limiting features for each build, or best coding practices. A recent Gartner survey showed that the top three priorities for development teams were as follows:

- Develop applications faster
- Expand the use of agile processes
- Reduce application-development costs

While performance was on the list it was down near the bottom. I find this ironic because in order to address the top three priorities performance should be the foundational priority. Let's look at each of the top three priorities from the survey in the context of having performance as the cornerstone.

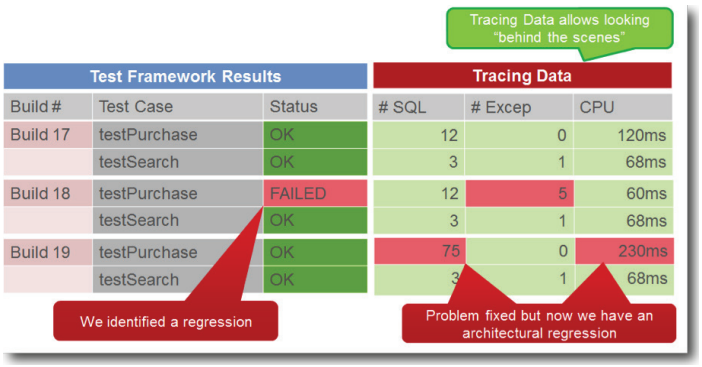
DEVELOP APPLICATIONS FASTER

With the advancement of web-delivered applications, companies are demanding increased interactions with customers and partners. The mobile revolution only compounds this. New features brought to market quickly can be the difference for a company. The agile process promotes quick iterations with small changes. This allows teams to churn out features rapidly. The drawback is that many times, in order to deliver applications on time, hard choices are made. Features are pushed and testing is seen as a luxury. As changes stack up problems that were introduced in previous sprints will become harder to unwind. This can come back to inhibit the scale of an application. A problem can be so embedded that the process to address it means tearing down several layers of functionality and rebuilding. Velocity stalls as teams must stop innovating to deal with bottlenecks that were introduced earlier in the development phase. The following graph visualizes what happens when performance is not treated as a high priority:



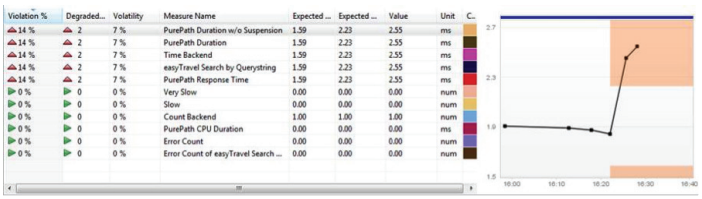
Performance must be a focus throughout development to avoid missed goals and missed expectations.

Let's look at how having performance as the foundation can impact faster releases. Using a performance platform earlier in the process gives developers insight into the performance of new application features. In addition to looking at test results from unit and integration tests, it is important to look at performance metrics that can be extracted from these test runs. This allows verification of functionality as well as architectural and performance problems by leveraging existing test assets. A best practice is to look at metrics such as *number of executed database statements*, *number of thrown exceptions*, *total execution time*, *created objects*, and *generated log statements*. The following table highlights some of these metrics to look at:



Analyzing performance Indicators for every test run identifies problems as they get introduced in the code.

Each code change is now tested for functional viability (*unit test status*), architectural correctness (*number of SQL statements*, *number of exceptions*, etc.) and performance (*execution time*, etc.). This indicates how the changes are impacting the overall quality and performance of an application. The following figure shows a different representation of this data, with built-in regression detection. The performance-management platform learns the expected value ranges for each performance indicator. If a metric is out of range, the platform automatically alerts on this regression:



Performance indicators that are out of the expected value range will automatically trigger alerts.

Developers see how the latest changes impact performance of a particular component that got tested. Each test can now generate a new task to address these automatically identified problems. The additional information that got captured in order to calculate these indicators helps when analyzing the actual regression. The following screenshot shows the comparison between the "last known good" and the problematic transaction that raised the alert.

Visual comparison makes it easy to identify the actual regression.

There is no need to write performance-specific user stories, as the platform automatically detects the test platform—in this case JUnit—and gives performance feedback for that build. Testing of performance becomes part of the automation process. With each build in the sprint performance is measured and compared to all pervious builds. It now just becomes a part of the sprint's overall "doneness."

An agile team can eliminate some of the stabilization sprints that are needed as performance defects have been vetted out in line. This keeps innovation moving and creates fewer chances for velocity to stall out due to stability issues.

EXPAND THE USE OF AGILE PROCESSES

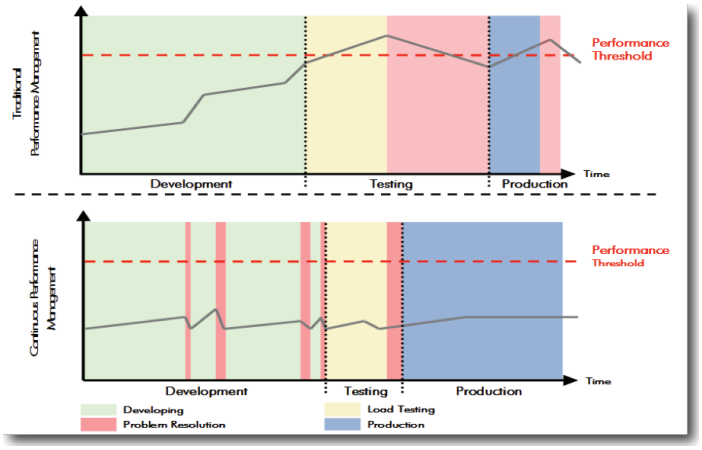
It is not a surprise that this is a top concern for development teams. More companies are moving from a waterfall structure to an agile process. The concept of test automation must be implemented for this transition to be successful. This is a time-consuming process, as most agile teams are not starting from scratch, but rather transitioning an application. Without these tests in place a core feature of the agile practice is lost. A main factor in regard to the adoption of agile practices and test automation is visibility. If no one is looking at the data from build to build then there is no reason to have the tests in the first place. Developers will focus on what is important to the completion of the project, and writing tests takes precious time and resources. This is a problem if you want to expand the use of agile practices.

With a solution that is integrated into the functional testing process, you have instance feedback about each test. Now the data provided is much richer than a simple pass/fail of tests. There is now an initial performance indication for each test. Teams are looking at this data for every build and visibility is high around it. There is a need to have coverage across the code base, as performance is now a critical component in the development stage. This need for understanding performance drives the adoption of test automation for agile teams, engraining it as standard practice.

REDUCE APPLICATION-DEVELOPMENT COSTS

This is just a reality of the world we live in now. For a development team time truly equals money. One developer costs x dollars per hour. In order to reduce the cost of development you must reduce the amount of time needed to deliver new features. This can be achieved by reducing the number of iterations needed to complete a new feature. Currently most performance is assessed at the testing level. As stated earlier, performance issues could be layered into several sprints that appear only in large-scale testing. This can set back a team several sprints, only adding to the cost.

Understanding performance earlier in the development cycle cuts down on the number of iterations needed to stabilize the code base for a new release. This allows for teams to keep the velocity of development very efficient and clean. The following charts illustrate this by comparing traditional performance management (performance as an after-thought) and continuous performance management (performance as a top priority).



Continuously focusing on performance results in better quality and on-time delivery.

Development is where ideas start to become reality. This is where applications specs begin to take shape and logical processes begin to morph into user interfaces and interactions. As you can see, the need to have performance in the development part of the lifecycle really is the cornerstone for an application. The slightest deviation can cause a ripple that may not appear until later in the lifecycle. So when we look at performance in this part of the lifecycle, it is surprising how little is done. In order to really become proactive, performance must be addressed from the start. The only way to do that is to have a platform in place that spans the lifecycle. In a future article I will move along the process and address how the acceptance team is crucial in managing performance and is a key component in becoming truly proactive.

If you are a dynaTrace user check out the following article on the Community Portal: *dynaTrace in Continuous Integration—The Big Picture*:
https://apmcommunity.compuware.com/community/display/PUB/dynaTrace+in+Continuous+Integration+-+The+Big+Picture

HOW TO MAKE DEVELOPERS WRITE PERFORMANCE TESTS

— Alois Reitbauer, March 11, 2011

I had an interesting conversation with our Test Automation team lead, Stefan (who Andi interviewed for our Eating Our Own Dog Food—**http://apmblog.compuware.com/2010/01/27/eating-our-own-dog-food-how-dynatrace-does-continuous-apm-internally-in-development-with-dynatrace-ta-lead-stefan-frandl/** article) on his experiences with the willingness of developers to write performance tests.

I asked a provocative question: do developers really want to write them in the first place? He smiled, but then he said that they do. I was a bit surprised because my own experience as a developer was that I wanted to write code instead of tests.

He agreed that it was not that easy in the beginning; however, after the first developers started it became a more and more common practice. I then asked him what he thinks are the main reasons why developers want to write performance tests. The following are three reasons for developers themselves to want to write performance tests.

REASON 1: EVERYONE WANTS TO BE A HERO

Developers like to write well-performing code. Even more, they like to write faster code. It's like Captain Kirk saying, "I need the ship ready in two hours" and Scotty replying, "I will do it in 20 minutes."

Every developer wants to show how well his code performs. The performance-testing hype at dynaTrace really started when a developer sent out an email about how his performance tests showed him a regression immediately after checking in. The figure below shows exactly the image the developer sent out.

Continuous-improvement performance graph showing several performance regressions

The developer wanted to share his experience and show others the benefits of the tests he wrote. A couple of weeks after this email, another developer sent out an email about the improvements he achieved for a component between two releases.

Another team of developers managed to optimize the throughput of our server by up to four times. This was a really great achievement and the result of hard work—earning them the dynaTrace culture award. Their contribution was acknowledged as one of the most significant in the last quarter.

If you want people to deliver certain results, reward them for those results and give them the fame they deserve.

REASON 2: THE "IT WILL COME BACK ANYWAY" EXPERIENCE

This one is nice, as nobody believes it at first. You can always tell people that if they do not get things right during development, a potential problem will come back to them later. So while performance testing might mean some additional work right now, it is better than having even more work later on when you have to fix performance bugs.

As I said, nobody believes this until they experience it themselves. Then people start to write their tests to avoid running into this situation again. There is nothing more frustrating than fixing bugs—possibly in older versions—when all your colleagues are working on the cool new features.

REASON 3: HE IS WORKING ON MY CODE TOO

At dynaTrace we have always taken performance seriously. Developers have always profiled their code to optimize it or add additional output for performance measurement. This works fine if you are the only one working on your code and do not have any dependencies on other components. If you not earning money by writing "Hello World" applications, though, this is not the case! Other developers are working on the code base as well.

So beside the fact that your profiling results are only available to you locally, they might already be outdated after the next check-in. As you have no automated means to find those changes, you will not be aware of them. Only a standardized test suite that is automatically executed as part of your continuous-integration environment can help here.

Some people also use specific logging code, which they think will help them to track down problems. While this works fine sometimes, our experience has shown that others might change the code, causing part of the logging statement to not be executed anymore. The reason might be that your code just got deprecated or somebody added an additional conditional statement, so logging code now gets executed in only certain cases.

REASON 4: LET DEVELOPERS WRITE ONLY TEST CODE

This is a key lesson we learned in making performance testing successful. The more complex it is to write test code, the fewer tests are written. Performance tests can easily become quite complex—sometimes even more complex than the actual code you want to test. In our case, we want to start our own software, start a couple of application-server instances, and then trigger a load test. All this has to be managed across multiple nodes.

If we forced developers to write that code themselves, they would not want to write any tests at all. So we made a massive investment in writing our own testing framework that takes away that burden from the developer. The example below shows how to start a remote WebSphere server by just adding some simple annotations.

```
@DtdRemoteSud(
host = “lab2”,
name = “WAS7.0”,
startupPriority = 1,
postStartClosure = WaitForWebSphereSudIsUp.class
)
private SudInterface webSphereSud;
```

Additionally, dynaTrace itself provides extensions developers can use to control our own software components. Developers now simply focus on writing their test cases. All this helps us to create a good test suite quickly and makes it attractive to developers to write tests.

REASON 5: PROVIDE FEEDBACK

If you invest your time, you want to get something back. So the key to getting developers to write more tests is to make the test results easily accessible to them. The immediate value for developers is to get feedback about the performance impact of their code changes very quickly. We run our performance-test suite two times a day. So ideally you can fix a performance bug the same day you created it.

Developers get this feedback via email just as build or functional-test results. So they invest time once and get feedback anytime they—or somebody else—changes the code.

REASON 6: THE OLD-CODE THREAT

Code ages, just as we do. The older code is, the less you want to touch it. However, very often you have no choice. Martin Fowler wrote in his Refactoring book that you start refactoring by having the proper tests in place. Martin was talking about functional tests, but the same is true for performance tests.

The really evil thing is that if you have modifying components that heavily interact with each other, you can very easily introduce performance problems. So instead of ruining just your own code, you will also ruin the code of other developers.

In particular, we have seen bug fixes introducing such problems. As you can see in the picture above, a functional bug fix can introduce a serious performance problem. Additionally, you want to be fast in fixing the old code so you have more time to develop the really cool new features—don’t you?

REASON 7: I MIGHT BE THE NEWB, BUT NOT THE NOOB

Stefan realized that hen the most test cases are written when a new developer takes over a feature. This is for two main reasons: First, you do not want to break other developers’ features. Second, you are new and want to learn how the code works. The best way to do this—except for reading source code and asking former developers—is by testing. So developers who take over a feature have the highest acceptance of writing tests.

YOUR WEB LOAD–TESTING QUESTIONS ANSWERED: PART ONE

— *Scott Barber,* Novevember 11, 2011

Scott Barber is the founder and chief technologist of PerfTestPlus, Inc., and is among the world's most prominent thought leaders in the area of software-systems performance testing. He is also a respected leader in advancing the understanding and practice of testing software systems, and is the author of *Web Load Testing for Dummies*, **http://offers2.compuware.com/apm_en_eb_Load_Testing_For_Dummies.html.** Here, Scott answers two big web load–testing questions.

Q: When gathering a project team to determine the goals of a performance-testing effort, what if the members of the team you gather know nothing about performance testing? Are there any tips to help get needed information from them?

SB: First, you want them to tell you what they want to learn as a result of doing performance testing qualitatively, not quantitatively. For example,

- Will this version of the site be faster than the last version?
- Will it be faster than our hated competitor's site?
- Will we be the ones in the news on Black Friday because our site crumbled under the load?
- Will our users complain when they try to use our app on a 3G network?
- Can we set up alerts in production to give us an early warning before it starts to slow?
- Can you tell me what component we'll need to upgrade first if we start getting a lot more traffic?
- Can we help the developers optimize their code early so we don't have a problem like last time?
- Will our current production hardware support this app at the projected volume? If not, how much more do I need to buy?

If you can collect questions like this, you can convert them into goals by turning each question into a statement, like this:

- Speed goal: Faster than hated competitor X
- Volume goal: Withstand Black Friday load

As for putting numbers to these goals, it's just a matter of doing your homework. For almost all of these, a few minutes with your favorite search engine, or a coffee and doughnut for your favorite IT/Ops member, and you'll have your numbers.

The only number that is actually difficult to come up with is target load. For that, you're going to have to listen to what your stakeholders are predicting (in terms of sales or total site members or searches or posts—whatever metric the sales team uses in reports to the CEO) and work backwards to come up with a number that makes sense. Don't be afraid to get some help, and don't be afraid to use the resources I mention below to help you out.

Q: We need to search for baseline using the current application measures, right? If we have a new application, it obviously will have neither historical data nor sales projections, and so on. In that case, how do we decide on the target load? Apart from looking at historical data of similar applications, is there any other way?

SB: If there is a previous version of the application, collecting performance-related data from both the production environment and the final set of performance tests you conducted on that version can be very useful. When you have a goal of performance that is no worse than the previous version (a baseline), there are plenty of other methods you can use to quantify your performance targets.

The easiest thing to do is to have key stakeholders refer you to a site or an application that exhibits performance characteristics they like. From there, you can easily paste the URL into a free online analyzer (like **http://www.websiteoptimization.com/services/analyze/**) or install a browser plug-in like Yahoo!'s YSlow or Google's PageSpeed (**http://developer.yahoo.com/yslow/ or https://developers.google.com/speed/pagespeed/**) and simply navigate to the site and view the analysis.

If you are interested in volume instead of speed, you can type your URL into **http://siteanalytics.compete.com/**, or even compare traffic volume across sites with **http://www.statsaholic.com/**. (I'm not specifically promoting these sites over others like them, these are simply the first ones that came to mind. These free tools are designed for websites. If you're not testing a website, use your favorite search engine to find free analyzers for the type of application you're interested in. Lots of companies make the basic functions of their analyzers available for free, or have free trial periods.)

Of course, these numbers won't be perfect, but neither will your historical data if you are making any user-noticeable changes to the site or application.

The reality is that you don't actually need to be all that close on your target load. What you need is a number to work toward. The system isn't going to handle that number (whatever it is) when you start running your tests—it's probably not going to handle that number when the stakeholders decide to ship it. And besides, odds are that you're not going to be testing on the production system anyway, so whatever number you get up to in your test environment is going to be different than what you're going to see in production anyway.

So make your best guess, and the less confident you are about the accuracy of that number, the bigger number you multiply that best guess by. As long as you have a number that is between accurate and greater than accurate that the team agrees is the target number (until the *team* agrees to change the target number to something else), then you can stop worrying about the number, start testing, and start worrying about all the performance issues keeping you from achieving that number.

Notice that not in *Web Load Testing for Dummies*, nor in this article, nor in anything I have written since 2003 do I use the phrase "performance requirement" except when there is a number defined in a legally enforceable contract, or when a number must be achieved or real human beings will be injured or die (for instance, the response time of an antilock-braking skid sensor).

Everything else is a *goal*, a target, or what I like to call *desirements*, because when it comes to performance, we all know that the stakeholders are going to decide to ship when they decide they are losing more money by not shipping than they think they will lose by shipping with whatever the performance numbers are at that point. After that, it's a monitoring and tuning issue. If you've done your job well, you'll be able to tell the folks doing the monitoring to keep an eye out so they can see the big slowdowns coming in advance and be proactive in minimizing the impact.

Read the rest of the interview with Scott Barber:

Part Two, **http://apmblog.compuware.com/2011/11/23/your-web-load-testing-questions-answered-part-two/**

Part Three, **http://apmblog.compuware.com/2011/12/22/load-testing-questions-answered-part-3/**

Part Four, **http://apmblog.compuware.com/2011/12/30/load-testing-questions-answered-part-4/**

THE TRUE VALUE OF APPLICATION PERFORMANCE MANAGEMENT IN PRODUCTION

— Michael Kopp, September 11, 2011

Often performance management is confused with performance troubleshooting. Additionally, many people think performance management in production is simply about system- and JVM-level monitoring and that they are already doing application performance management (APM).

The first perception assumes that APM is about speeding up some arbitrary method performance, and the second assumes that performance management is just about discovering that something is slow. Neither of these two is what we at dynaTrace would consider prime drivers for APM in production. So what does it mean to have APM in production, and why do you do it?

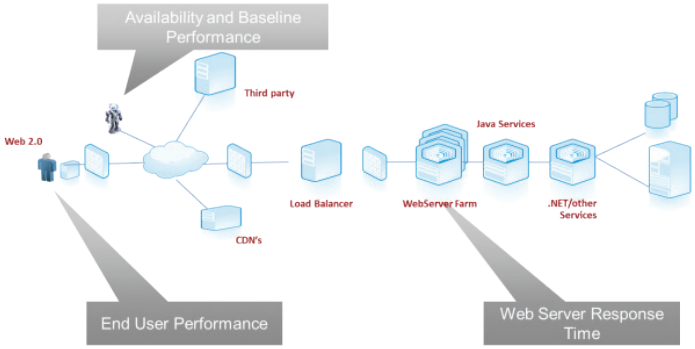
The reason our customers need APM in their production systems is to understand the impact that end-to-end performance has on their end users and therefore their business. Our customers use this information to optimize and fix their applications in a way that has direct and measurable return on investment (ROI). This might sound easy, but in environments that include literally thousands of JVMs and millions of transactions per hour, nothing is easy unless you have the right approach!

Therefore, real APM in production solves problems such as the following:

- Reveals how performance affects end users' buying behavior or the revenue of your tenants
- Monitors the performance of your search for a specific category
- Lets you know which of your 100 JVMs, 30 C++ business components, and three databases is participating in your transaction, and which of them is responsible for your problem
- Enables Operations, Business, and R&D to look at the same production performance data from their respective vantage points
- Enables R&D to analyze production-level data without requiring access to the production system

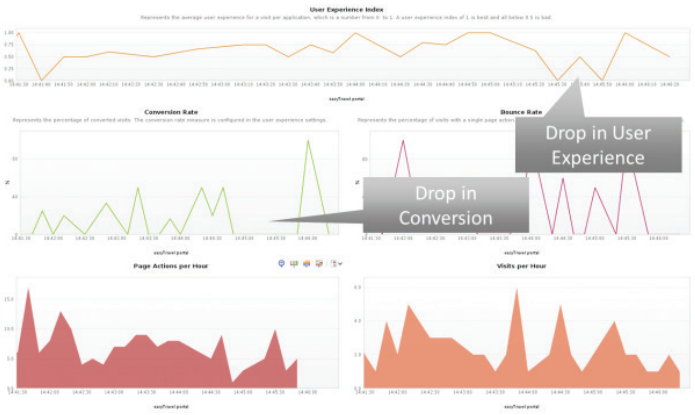
GAIN END-TO-END VISIBILITY

The first thing you realize when looking at any serious web application—for instance, any of the big ecommerce sites—is that much of the end-user response time gets spent outside the company's datacenter. Doing performance management on only the server side leaves you blind to problems caused due to JavaScript, content-delivery networks, third parties, and, in the case of mobile users, simple bandwidth.



Web delivery chain

As you are not even aware of these problems, you cannot fix them. Without knowing the effect that performance has on your users, you do not know how performance impacts your business. Without knowing that, how do you decide if your performance is OK?

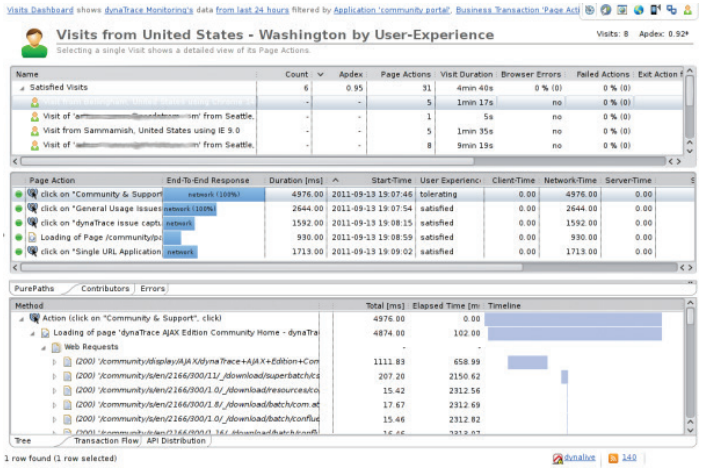


This dashboard shows that there is a relationship between performance and conversion rate.

The primary metric at the end-user level is the conversion rate. End-to-end APM tells you how application performance or nonperformance impacts that rate. In other words, you can put a dollar value on response time and error rate!

Thus, the first reason why you do APM in production is to understand the impact that performance and errors have on your users' behavior.

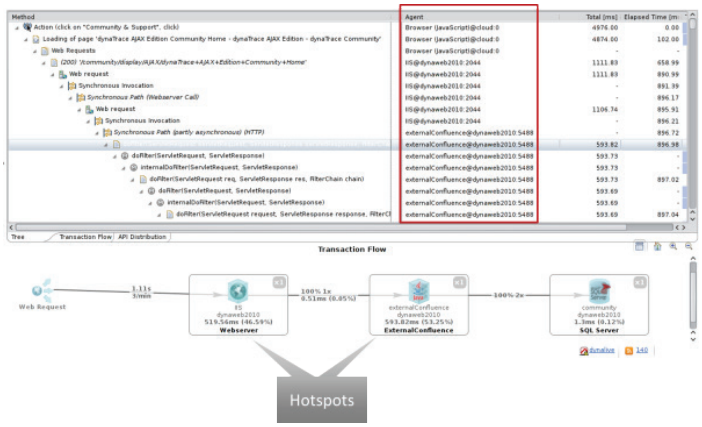
Once you know the impact that some slow request has on your business, you want to zero in on the root cause, which can be anywhere in the web delivery chain. If your issue is on the browser side, you want the exact click path of the affected users.



A visit's click path plus the Page Action PurePath of the first click

You can use this to figure out if the issue is in a specific server-side request, is related to third-party requests, or is in the JavaScript code. Once you have the click path, plus some additional context information, a developer can easily use something like AJAX Edition, **http://www.compuware.com/application-performance-management/ajax-performance-testing.html**, to analyze it.

If the issue is on the server side, you need to isolate the root cause there. Many environments today encompass several hundred JVMs, common-language runtimes (CLRs), and other components. They are big, distributed, and heterogeneous. To isolate a root cause here you need to be able to extend the click path into the server itself.



From the click path to the server side

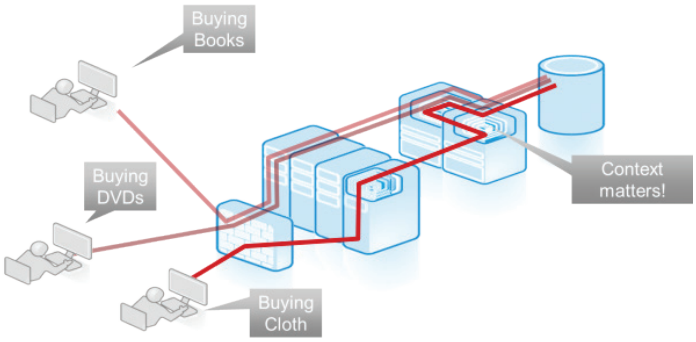
But before we look at that, we should look at the other main driver of performance management—the business itself.

CREATE FOCUS—IT'S THE BUSINESS THAT MATTERS

Old forms of performance management are disconnected from the business. It simply has no meaning for the business to know that average CPU on 100 servers is at 70% (or whatever else). It does not mean anything to say that JBoss xyz has a response time of 1 second on web page abc. Is that good or bad? Why should I invest money to improve that? On top of this, we don't have one server, but rather thousands of them, with thousands of different web pages and services all calling each other, so where should we start? How do we even know if we should do something?

The last question is crucial and is the second main reason why we do APM. We combine end-user monitoring, **http://www.compuware.com/application-performance-management/use-case-user-experience-management.html**, with business-transaction management, **http://www.compuware.com/application-performance-management/use-case-business-transaction-management.html**. We want to know the impact performance has on our business and whether the business performance of our services is influenced by performance problems of our applications.

While end-user monitoring enables you to put a general dollar figure on your end-user performance, business transactions go one step further. Let's assume the user can buy different products based on categories. If I have a performance issue, I would want to know how it affects my best-selling categories and I would prioritize based on that. The different product categories trigger different services on the server side. This is important for performance management in itself, as I would otherwise look at too much data and could not focus on what matters.



The payment transaction's path depends on context.

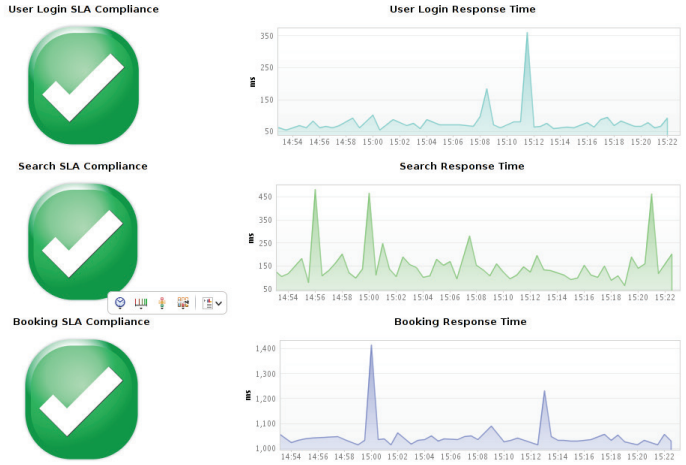
Business-transaction management does not just label a specific web request with a name, such as Booking for a booking transaction, but rather enables you to do performance management on a higher level. It is about knowing if and why revenue of one tenant is affected by the response time of the booking transaction.

In this way business transactions create a twofold focus. They enable the business side and management to set the right focus. That focus is always based on company success, revenue, and ROI. At the same time, business transactions enable developers to exclude 90% of the noise from their investigation and immediately zero in on the real root cause. This is due to the additional context that business-transaction management brings. For instance, if only bookings via credit cards are affected, then diagnostics should focus on only these and not all booking transactions. This brings me to the diagnosis of performance issues in production.

THE PERFECT STORM OF COMPLEXITY

At dynaTrace we regularly see environments with several hundred or even more than a thousand web servers, JVMs, CLRs, and other components running as part of a single application environment. These environments are not homogeneous. They include native business components; integrations with, for example, Siebel or SAP; and, of course, the mainframe. These systems are here to stay, and their impact on the complexity of today's environments cannot be underestimated. Mastering this complexity is another reason for APM.

Today's systems serve huge user bases, and in some cases need to process millions of transactions per hour. Ironically, most APM solutions and approaches will simply break down in such an environment, but the value that the right APM approach brings is vital. The way to master such an environment is to look at it from an application and transaction point of view.



Monitoring of service-level agreements (SLAs)

SLA violations and errors need to be detected automatically, **http://apmblog.compuware.com/2011/09/13/automatic-error-detection-in-production-contact-your-users-before-they-contact-you/** and the data to investigate needs to be captured; otherwise we will never have the ability to fix it. The first step is to isolate the offending tier and find out if the problem is due to the host, the database, the JVM, the mainframe, a third-party service, or the application itself.

Isolating the credit-card tier as the root cause

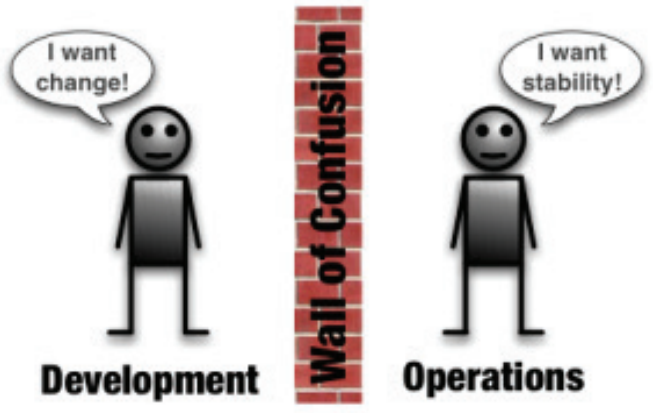
Instead of seeing hundreds of servers and millions of data points, we can immediately isolate the one or two components that are responsible for the issue. Issues happening here cannot be reproduced in a test setup. This has nothing to do with lack of technical ability; we simply do not have the time to figure out which circumstances lead to a problem. So we need to ensure that we have all the data we need for later analysis available all the time. This is another reason why we do APM. It gives us the ability to diagnose and understand real-world issues.

Once we have identified the offending tier, we know who to talk to. That brings me to my last point—collaboration.

BREAKING THE LANGUAGE BARRIER

Operations is looking at SLA violations and uptime of services; the business side is looking at revenue statistics of sold products; and R&D is thinking in terms of response time, CPU cycles, and garbage collection. It is a fact that these three teams speak completely different languages. APM is about presenting the same data in those different languages, and thus breaking the barrier.

Another factor is that as a developer you never get access to the production environment, so you have a hard time analyzing the issues. Reproducing issues in a test setup is often not possible either. Even if you were to have access, most issues cannot be analyzed in real time. In order to effectively share the performance data with R&D, we first need to capture and persist it. It is important to capture all transactions and not just a subset. Some think you need to capture only slow transactions, but there are several problems with this. Either you need to define what is slow or, if you have been baselining, you must note what is slower than before. The first is a lot of work, and the second assumes that performance is fine right now. That is not good enough. In addition, such an approach ignores the fact that concurrency exists. Concurrently running transactions impact each other in numerous ways, and whoever diagnoses an issue at hand will need that additional context.



A typical Operations-to-Development conversation without APM

Once you have the data, you need to share it with R&D, which most of the time means to physically copy a scrubbed version of that data to the R&D team. While the scrubbed data must exclude things like credit-card numbers, it must not lose its integrity. Developers need to be able to look at exactly the same picture as Operations. This enables better communication with Operations while enabling deep-dive diagnostics.

Once a fix has been supplied, Operations needs to ensure that there are no negative side effects and will verify that it has the desired positive effect. Modern APM solves this by automatically understanding the dynamic dependencies between applications and automatically monitoring new code for performance degradations.

Thus, using APM in production improves communication, speeds up deployment cycles, and adds a layer of quality assurance. This is the final—but by far not least important—reason we do APM.

CONCLUSION

The reason we do APM in production is not to fix a CPU hot spot, speed up a specific algorithm, or improve garbage collection. Neither the business side nor Operations cares about that. We do APM to understand the impact that an application's performance has on our customers and thus our business. This enables us to effectively invest precious development time where it has the most impact, thereby furthering the success of the company. APM truly serves the business needs of a company and its customers by bringing focus to the performance-management discipline.

If you do APM in production, and you should, do it for the right reasons.

PLANNING FOR HOLIDAY WEB LOAD–TESTING

— Laura Strassman, October 4, 2012

Every year, the arrival of autumn signals that it's time for holiday website-traffic planning. Here I'll provide some insights and web load–testing best practices for that busy season. There are four main areas of focus:

1. Dedicate time to planning

You shouldn't just wake up one morning and decide to web load–test your site. You want to plan it out so that you test the right things, and if your testing cycles are short or not as extensive as you hoped, you test what is most important.

2. Get the whole team on board

By the whole team I don't mean all of your QA test engineers. I mean everyone with a stake in the holiday website. The list should include a representative from Marketing, IT operations, Development, possibly a business manager or PR person, and, of course, the QA team.

Both the business side and IT must provide input into the testing goals and success criteria, as well as the test plan and methodology. The business team is in the best position to estimate the expected traffic and sales growth, as well as provide campaign information. The IT team needs to ensure that the business side understands the capabilities and limitations of the environment and the level of effort and cost to meet the growth demands.

Getting the whole team in a room to hash out what has changed on the website, and what the expectations are for revenue generation, traffic spikes, and customer performance, is how you move on to the third area of focus.

3. Establish goals

Each team member from each department will have goals for the web application. Additionally, you have to think about your customers' goals so you can test from their perspective. That is how you establish your testing goals.

As you work to set your testing goals, consider the following:

- Will the site break under anticipated peak load? Can we handle unanticipated surges in traffic?
- Will our most important transactions and pages perform properly, regardless of load?
- Will our customers have a positive experience no matter where they are located geographically or what device they may be using?
- Are our site's third-party providers (of rating and reviews, ads, news feeds, ecommerce engines, or content-delivery networks) hurting our performance?

4. Figure out what to measure

Based on the team's input, you can narrow your focus to what is important to test—what metrics you need to obtain. The following are some common measurements:

- User-perceived response time
- Response time by geography
- Number of virtual users—measured at many points in time
- Resource utilization (such as CPU memory, disk I/O, and network bandwidth)
- Number of errors

Approaching your testing in this planned manner ensures that the right input is received so that the output means something. In our experience helping customers prepare for the holiday season, we have witnessed IT teams moving forward with load testing without direct involvement from the business side, whether due to corporate culture, lack of knowledge, or perceived lack of time. This isolated approach causes unnecessary stress, and any time saved by not have a couple of meetings is easily wiped out if the testing doesn't test the right things. (Tests may succeed in a vacuum, but until the site is launched and the volume of users peaks, you have no way of knowing if you really have a site that will support your users.)

For more insight into preparing for holiday web load–testing, watch this short video: Load Testing Tips for the Holidays—**http://www.youtube.com/watch?v=BZsDAN90V4c&feature=youtu.be**.

For more industry-leading Application Performance Management articles, read the about:performance blog at: **apmblog.compuware.com**

Compuware Corporation, the technology performance company, provides software, experts and best practices to ensure technology works well and delivers value. Compuware solutions make the world's most important technologies perform at their best for leading organizations worldwide, including 46 of the top 50 Fortune 500 companies and 12 of the top 20 most visited U.S. web sites. Learn more at: **compuware.com**.

Compuware Corporation World Headquarters • One Campus Martius • Detroit, MI 48226-5099

Compuware

THE TECHNOLOGY PERFORMANCE COMPANY